CS 416 - Operating Systems Design

Homework 3
Virtual Memory in xv6

November 21, 2016
Due by Friday, December 9th at 11:55PM

1 Introduction

In this homework, you will explore the xv6 virtual memory systems, as well as expand the virtual memory-based features of the operating system. In particular, you will implement a memory protection facility and copy-on-write for fork.

2 Implementing Memory Protection

2.1 Requirements

Implement an `mprotect()` system call in xv6 similar to the one available on Linux. The method signature should be as follows (*note that **addr** *must* be aligned to a page boundary*):

```
int mprotect(void *addr, int len, int prot)
```

You should implement support for the following protection levels (there is no need to implement `PROT_EXEC`).

- `PROT_NONE` - The memory cannot be accessed at all.
- `PROT_READ` - The memory can be read.
- `PROT_WRITE` - The memory can be modified.

See the Linux man page for the details of how `mprotect()` should behave.

When a process violates memory protection a signal called `SIGSEGV` should occur. You will need to modify the provided signal implementation similar to your first homework to add the `SIGSEGV` signal, as well as add fields to your `siginfo_t` struct. In particular, `siginfo_t` must contain information about the nature of the protection fault (the address and type of exception).

The `siginfo_t` should include the following fields:

```
uint addr; // Should be an address.
uint type; // Should be a protection level.
```

For preexisting exceptions, these fields can simply contain zeroes, as we can consider those values undefined for that signal.

2.1.1 Signal Starting Point

The starting branch will contain a functional implementation of the signal facility similar to Homework 1, in order to let you focus on the memory protection aspects, rather than the signal facility. However, you'll need to make some changes to this implementation to meet the new signal handling requirements mentioned in the previous section.

2.2 Testing

A test program called **test_mprotect** is in the branch that will test your new memory protection facility. The program **test_mprotect** will (at a minimum) do the following:

- Allocate a page.
- Mark a page read only using **mprotect()**
- Attempt to write to the page.
- Handle **SIGSEGV**.
- Use **mprotect()** inside the handler to mark the page read-write.
- Print all addresses and the status at each step.

This program is commented out in the makefile. You should uncomment it once you have updated your signal handling facility and have implemented the **mprotect** system call.

3 Implementing Copy On Write

3.1 Requirements

Implement copy-on-write for process images. When **fork()** occurs in the normal case, the process image is copied to the new child process before the child starts executing. You should implement a new system call **cowfork()** that functions identically to the normal **fork()**, except that instead of copying the process image, the mappings should instead be duplicated and marked read-only in both processes. To keep track of this, the kernel should be able to track the number of these read-only mappings to each page. When a write is attempted into any read-only page, the page should be copied into a new read-write page, the mapping updated, and the counter decremented on the original page. You will need to be very careful on **exec()** and **exit()**, when all of the mappings are destroyed and the and counters decremented.

3.2 Testing

The starting branch will contain a test program called **test_cow** to test your copy on write facility. The program test cow will do the following:

- Make a large number of calls to **fork()** and calculate the average time for each call.
- Make the same number of calls to **cowfork()**, and calculate the average time for each call.
- Cleans up the child processes after the test.

4 Source Control

To start on this project, use the following commands in your xv6 git repository from Homework 2.

```
git checkout fa16-hw3-start
```

5 Submission

To submit your work, please tar up your entire xv6 repository and submit this tarball on Sakai.

Only ONE student from each group should commit, but you NEED to make sure you give the netids of both partners. Note that group members cannot change from the ones in homework 2.

6 Overall Requirements

- The code has to be written in **C** language. You should discuss on Sakai if you think inline **asm** is necessary to accomplish something.
- Do not copy the solution from other students. Use Sakai to discuss ideas of how to implement it. Do not post your solution there.
- Commit in your local git for each step of your solution to make sure you can isolate each step later in case you need to refer back.
- Submit a report on **Sakai** named **report.pdf**, detailing what you accomplished for this project, including issues you encountered. This report *must* be named **report.pdf** and *must* be in PDF format.